

**ScreenRecorder: A Utility for Creating Screenshot Video
Using Only Original Equipment Manufacturer (OEM)
Software on Microsoft Windows Systems**

by Mary K Arthur

ARL-TN-0658

January 2015

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5069

ARL-TN-0658**January 2015**

ScreenRecorder: A Utility for Creating Screenshot Video Using Only Original Equipment Manufacturer (OEM) Software on Microsoft Windows Systems

Mary K Arthur

Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) January 2015		2. REPORT TYPE Final		3. DATES COVERED (From - To) August 2014	
4. TITLE AND SUBTITLE ScreenRecorder: A Utility for Creating Screenshot Video Using Only Original Equipment Manufacturer (OEM) Software on Microsoft Windows Systems				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Mary K Arthur				5d. PROJECT NUMBER AH80	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-WML-A Aberdeen Proving Ground, MD 21005-5069				8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TN-0658	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Being able to save screenshots directly to a file and to record activity on a desktop or within a specific window is extremely useful for a variety of reasons. Although third-party screen-recording software is readily available online from a variety of sources, such software often presents a security concern and would require administrative permissions to install. This report presents a software utility for capturing series of desktop or window screenshots using only original equipment manufacturer software on systems running Microsoft Windows.					
15. SUBJECT TERMS screenshot, screen capture, screen record, desktop, window, handle, Microsoft, Windows, original equipment manufacturer, OEM, Visual Studio, Movie Maker, C++, object-oriented					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 46	19a. NAME OF RESPONSIBLE PERSON Mary K Arthur
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (Include area code) 410-278-6110

Contents

List of Figures	iv
Acknowledgments	v
1. Introduction	1
2. ScreenRecorder: Static Member Functions, Screen Capture	2
2.1 CaptureDesktop	2
2.2 CaptureByTitle	3
2.3 CaptureByHandle	5
3. ScreenRecorder: Class Variables, Constructors, Accessors, and Mutators	6
3.1 Constructors.....	6
3.2 Accessors.....	7
3.3 Mutators.....	7
4. ScreenRecorder: Non-Static Member Functions, Screen Record	8
4.1 RecordDesktop	8
4.2 RecordByTitle	9
4.3 RecordByHandle	11
5. Conclusion	12
6. References and Notes	13
Appendix A. ScreenRecorderExamples_Capture.cc	15
Appendix B. ScreenRecorderExamples_Record.cc	19
Appendix C. ScreenRecorder.h	23
Appendix D. ScreenRecorder.cc	27
Appendix E. MATLAB Script for Creating AVI Movie from Screenshots	35
Distribution List	37

List of Figures

Fig. 1	Static member function (screen capture) declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and Appendix D for full class implementation in ScreenRecorder.cc.	2
Fig. 2	Sample usage of CaptureDesktop functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.	3
Fig. 3	Sample usage of CaptureByTitle functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.	4
Fig. 4	Sample usage of CaptureByHandle functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.	5
Fig. 5	Constructor declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h, and appendix D for full class implementation in ScreenRecorder.cc.	6
Fig. 6	Accessor declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.	7
Fig. 7	Mutator declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.	7
Fig. 8	Non-static member function (screen record) declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.	8
Fig. 9	Sample usage of RecordDesktop functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.	9
Fig. 10	Sample usage of RecordByTitle functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.	10
Fig. 11	Sample usage of RecordByHandle functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.	11

Acknowledgments

I would like to thank Ben Breech for his extremely thorough technical review, feedback, and suggestions that greatly improved the quality of the ScreenRecorder utility and this report. I would also like to thank Mark Gatlin for his editorial review.

INTENTIONALLY LEFT BLANK.

1. Introduction

Being able to save screenshots directly to a file and to record activity on a desktop or within a specific window is extremely useful for a variety of reasons including, but not limited to, diagnosing computer problems, teaching and/or demonstrating software, and generating videos for presentations. Although third-party screen-recording software is readily available online from a variety of sources, such software often presents a security concern and would require administrative permissions to install. This report presents ScreenRecorder, a software utility for capturing series of desktop or window screenshots using only original equipment manufacturer (OEM) software on systems running Microsoft Windows.¹

The ScreenRecorder utility was developed as an objected-oriented C++ class within Microsoft Visual Studio.² It has been tested on and is compatible with Microsoft Vista, 7, and 8 and Visual Studio Express 2008, 2010, and 2012. The utility supports 5 image formats: Windows Bitmap (BMP), Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), and Tagged Image File Format (TIFF). Once the ScreenRecorder utility has saved a series of screenshots in the desired format, Windows Movie Maker³ can be used to combine the screenshot images into a Windows media file (WMV) or Audio-Video Interleaved (AVI) movie file. This report does not explicitly address creating a video from a series of images but rather directs the user to Microsoft's Windows Movie Maker tutorial⁴ for further information regarding creating videos from images using Microsoft OEM software.⁵

Functions within the ScreenRecorder utility fall into 3 categories: 1) static screen capture member functions for saving a single screenshot to a file that can be invoked without using a ScreenRecorder object, 2) functions for creating and manipulating ScreenRecorder objects (constructors, accessors, and mutators), and 3) non-static screen recording member functions for saving a series of screenshots to file that can only be accessed through a ScreenRecorder object.

The use of the ScreenRecorder utility assumes a basic understanding of compiling and running C++ code within Microsoft Visual Studio. This report does not attempt to explain the inner workings of the ScreenRecorder utility but is published as a means to distribute the source code and act as a user's manual. All source code is provided to the user within the appendices. The source code is completely self-contained, and the user need only include the files in a Visual Studio project and link to the following 3 standard Windows libraries: user32, gdi32, and gdiplus. See example codes in Appendices A and B for an example of how to link these libraries.

2. ScreenRecorder: Static Member Functions, Screen Capture

The ScreenRecorder utility includes 4 static member functions for capturing a single screenshot image: CaptureDesktop, CaptureByTitle, and 2 versions of CaptureByHandle (Fig. 1).

21	<code>// **** Static Methods - Screen Capture ****</code>
22	
23	<code>static bool CaptureDesktop(std::string saveAsFile = "ScreenCapture.png",</code>
24	<code>std::string saveAsPath = "");</code>
25	<code>static bool CaptureByTitle(std::string title,</code>
26	<code>std::string saveAsFile = "ScreenCapture.png",</code>
27	<code>std::string saveAsPath = "");</code>
28	<code>static bool CaptureByHandle(std::string handle,</code>
29	<code>std::string saveAsFile = "ScreenCapture.png",</code>
30	<code>std::string saveAsPath = "");</code>
31	<code>static bool CaptureByHandle(HWND hWnd,</code>
32	<code>std::string saveAsFile = "ScreenCapture.png",</code>
33	<code>std::string saveAsPath = "");</code>

Fig. 1 Static member function (screen capture) declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and Appendix D for full class implementation in ScreenRecorder.cc.

2.1 CaptureDesktop

The CaptureDesktop function saves a single image of the desktop. CaptureDesktop takes up to 2 string⁶ parameters.

If no parameters are supplied, the image of the desktop is saved as ScreenCapture.png in the same directory as the executable (Fig. 2, lines 31 and 32).

If a single string parameter is supplied, this parameter specifies the save-as filename. The image of the desktop is then saved with the specified filename in the same directory as the executable (Fig. 2, lines 37 and 38).

If 2 string parameters are supplied, the first parameter specifies the save-as filename and the second is the path to the desired save location. The image of the desktop is then saved with the specified filename in the specified directory (Fig. 2, lines 43–46).

```

26   ScreenRecorder recorder; // ScreenRecorder object
27
28   // **** ScreenRecorder::CaptureDesktop *****
29
30   // CaptureDesktop
31   if (!ScreenRecorder::CaptureDesktop() || // Invoked without use of a class object
32       !recorder.CaptureDesktop()) {        // Invoked through a class object
33       std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
34   } // if (!CaptureDesktop())
35
36   // CaptureDesktop(saveAsFile)
37   if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.bmp") ||
38       !recorder.CaptureDesktop("CaptureDesktop.bmp")) {
39       std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
40   } // if (!CaptureDesktop(...))
41
42   // CaptureDesktop(saveAsFile, saveAsPath)
43   if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.gif",
44                                         "C:\\Documents\\ScreenRecorder") ||
45       !recorder.CaptureDesktop("CaptureDesktop.gif",
46                                   "C:\\Documents\\ScreenRecorder")) {
47       std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
48   } // if (!CaptureDesktop(...))

```

Fig. 2 Sample usage of CaptureDesktop functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.

The return value of CaptureDesktop indicates whether or not an image of the desktop was successfully saved. If the user specifies the filename, the CaptureDesktop function will return false if the specified filename does not end with one of the supported file format extensions (i.e., bmp, gif, jpg/jpeg, png, or tif/tiff). If the user specifies a save path that does not exist, CaptureDesktop will return false.

Windows is generally case-insensitive; all functions in the ScreenRecorder utility are case-insensitive.

2.2 CaptureByTitle

The CaptureByTitle function saves a single image of the window identified by the specified title. CaptureByTitle requires at least one string parameter (the title of the window to capture) followed by up to 2 string parameters.

If only the required title string parameter is supplied, the image of the specified window is saved as ScreenCapture.png in the same directory as the executable (Fig. 3, lines 53 and 54).

```

26   ScreenRecorder recorder; // ScreenRecorder object

50   // **** ScreenRecorder::CaptureByTitle ****
51
52   // CaptureByTitle(windowTitle)
53   if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]") ||
54       !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]")) {
55       std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
56   } // if (!CaptureByTitle(...))
57
58   // CaptureByTitle(windowTitle, saveAsFile)
59   if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
60       "CaptureByTitle.jpeg") ||
61       !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
62       "CaptureByTitle.jpeg")) {
63       std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
64   } // if (!CaptureByTitle(...))
65
66   // CaptureByTitle(windowTitle, saveAsFile, saveAsPath)
67   if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
68       "CaptureByTitle.png",
69       "C:\\Documents\\ScreenRecorder") ||
70       !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
71       "CaptureByTitle.png",
72       "C:\\Documents\\ScreenRecorder")) {
73       std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
74   } // if (!CaptureByTitle(...))

```

Fig. 3 Sample usage of CaptureByTitle functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.

If a single string parameter is supplied in addition to the required title string parameter, this parameter specifies the save-as filename. The image of the specified window is then saved with the specified filename in the same directory as the executable (Fig. 3, lines 59–62).

If 2 string parameters are supplied in addition to the required title string parameter, the first parameter specifies the save-as filename and the second is the path to the desired save location. The image of the specified window is then saved with the specified filename in the specified directory (Fig. 3, lines 67–72).

The return value of CaptureByTitle indicates whether or not the specified window was found and an image of this window was successfully saved. If the user specifies the filename, the CaptureByTitle function will return false if the specified filename does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, CaptureByTitle will return false.

The “title” that appears at the top of a window (in the title bar) is not necessarily the actual title of the window. The easiest way to determine the actual title of the window is to open Windows Task Manager (ctrl + shift + esc), select the Applications tab, and identify the task that you are attempting to capture. The task names are the actual window titles.

2.3 CaptureByHandle

Both CaptureByHandle functions save a single image of the window identified by the specified window handle.⁷ Both CaptureByHandle functions require one parameter specifying the handle to the window to capture followed by up to 2 optional string parameters. The difference between the 2 CaptureByHandle functions is the data type of the required parameter specifying the handle to the window to capture. This parameter can either be a hex string identifier to the handle (e.g, “0012079E”) or a Microsoft Windows window handle (HWND).⁸

If only the required handle identifier parameter is supplied, the image of the specified window is saved as ScreenCapture.png in the same directory as the executable (Fig. 4, lines 85–88).

```
26   ScreenRecorder recorder; // ScreenRecorder object

76   // **** ScreenRecorder::CaptureByHandle ****
77
78   // Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
79   HWND      handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");
80
81   // Hex value of handleObject saved as a string
82   std::string handleString = toString(handleObject);
83
84   // CaptureByHandle(windowHandle)
85   if (!ScreenRecorder::CaptureByHandle(handleObject)           ||
86       !recorder.CaptureByHandle(handleObject)                 ||
87       !ScreenRecorder::CaptureByHandle(handleString)           ||
88       !recorder.CaptureByHandle(handleString)) {
89       std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
90   } // if (!CaptureByHandle(...))
91
92   // CaptureByHandle(windowHandle, saveAsFile)
93   if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.tiff") ||
94       !recorder.CaptureByHandle(handleObject, "CaptureByHandle.tif")         ||
95       !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.bmp")   ||
96       !recorder.CaptureByHandle(handleString, "CaptureByHandle.gif")) {
97       std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
98   } // if (!CaptureByHandle(...))
99
100  // CaptureByHandle(windowHandle, saveAsFile, saveAsPath)
101  if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.jpg",
102                                         "C:\\Documents\\ScreenRecorder") ||
103      !recorder.CaptureByHandle(handleObject, "CaptureByHandle.png",
104                                         "C:\\Documents\\ScreenRecorder") ||
105      !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.tif",
106                                         "C:\\Documents\\ScreenRecorder") ||
107      !recorder.CaptureByHandle(handleString, "CaptureByHandle.bmp",
108                                         "C:\\Documents\\ScreenRecorder")) {
109      std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
110  } // if (!CaptureByHandle(...))
```

Fig. 4 Sample usage of CaptureByHandle functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.

If a single string parameter is supplied in addition to the required handle identifier parameter, this parameter specifies the save-as filename. The image of the specified window is then saved with the specified filename in the same directory as the executable (Fig. 4, lines 93–96).

If 2 string parameters are supplied in addition to the required handle identifier parameter, the first parameter specifies the save-as filename and the second is the path to the desired save location. The image of the specified window is then saved with the specified filename in the specified directory (Fig. 4, lines 101–108).

The return value of `CaptureByHandle` indicates whether or not the specified window was found and an image of this window was successfully saved. If the user specifies the filename, the `CaptureByHandle` function will return false if the filename does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, `CaptureByHandle` will return false.

3. ScreenRecorder: Class Variables, Constructors, Accessors, and Mutators

A `ScreenRecorder` object has 3 class variables: record time, frame rate, and time step. The record-time variable is the duration of time in seconds over which the recorder object will capture a series of screenshots; it has type `double` and default value of 60.0 s. The frame-rate variable is the number of screenshots that will be captured each second; it has type `double` and default value of 1.0 frame per second (fps). The time-step variable is the time between screenshots in seconds; it has type `double` and default value of 1.0 s. The frame rate and time step are inversely proportional and related by the following equation:

$$timeStep = \frac{1}{frameRate}. \quad (1)$$

The `ScreenRecorder` utility includes 3 constructors, 3 accessor functions, and 3 mutator functions.

3.1 Constructors

The default constructor for a `ScreenRecorder` object does not take any parameters and sets the class variables to their default values (Fig. 5, line 37).

35 36 37 38 39	<pre> // **** Constructors **** ScreenRecorder(); ScreenRecorder(double recordTime, int frameRate); ScreenRecorder(double recordTime, double timeStep); </pre>
----------------------------	--

Fig. 5 Constructor declarations in `ScreenRecorder.h`. See Appendix C for full documentation of `ScreenRecorder.h`, and appendix D for full class implementation in `ScreenRecorder.cc`.

The remaining 2 constructors both require 2 parameters. The first parameter is the same for both constructors, has type double, and specifies the record time in seconds. The second parameter either has type integer and specifies the frame rate⁹ in frames per second (figure 5, line 38) or has type double and specifies the time step in seconds (figure 5, line 39). If a record time of less than 0.0 is specified, an error message is printed to the standard error stream (std::cerr) and the record time is set to the default 60.0 s. If a frame rate of 0 or less¹⁰ is specified, an error message is printed to std::cerr and the frame rate is set to the default 1.0 fps. If a time step of 0.0 or less is specified, an error message is printed to std::cerr and the time step is set to the default 1.0 s.

3.2 Accessors

Accessor functions, often called “getters”, are methods that return the state (value) of private member variables. The ScreenRecorder has an accessor for each of its 3 private member variables: GetRecordTime, GetFrameRate, and GetTimeStep (Fig. 6). Accessors do not take any parameters. GetRecordTime returns the duration of time in seconds over which the recorder object will capture a series of screenshots. GetFrameRate returns the frame rate of the recorder object in frames per second. GetTimeStep returns the time step of the recorder object in seconds.

41	// **** Accessors ****
42	
43	double GetRecordTime();
44	double GetFrameRate();
45	double GetTimeStep();

Fig. 6 Accessor declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.

3.3 Mutators

Mutator functions, often called “setters”, are methods for changing the state (value) of private member variables. The ScreenRecorder has a mutator for each of its 3 private member variables: SetRecordTime, SetFrameRate, and SetTimeStep (Fig. 7).

47	// **** Mutators ****
48	
49	void SetRecordTime(double recordTime);
50	void SetFrameRate(double frameRate);
51	void SetTimeStep(double dT);

Fig. 7 Mutator declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.

SetRecordTime takes one parameter of type double and with units in seconds and sets the duration of time over which the recorder object will capture a series of screenshots to this value. If a parameter value of less than 0.0 is passed to SetRecordTime, an error message is printed to std::cerr and the record time is set to the default 60.0 s.

SetFrameRate takes one parameter of type double and with units in frames per second and sets the frame rate of the recorder object to this value. If a parameter value that is less than or equal

to 0.0 is passed to `SetFrameRate`, an error message is printed to `std::cerr` and the frame rate is set to the default 1.0 fps. `SetFrameRate` also changes the object's time step to be equal to 1.0 divided by the new frame rate.

`SetTimeStep` takes one parameter of type `double` and with units in seconds and sets the time step of the recorder object to this value. If a parameter value that is less than or equal to 0.0 is passed to `SetTimeStep`, an error message is printed to `std::cerr` and the time step is set to the default 1.0 s. `SetTimeStep` also changes the object's frame rate to be equal to 1.0 divided by the new time step.

4. ScreenRecorder: Non-Static Member Functions, Screen Record

The `ScreenRecorder` utility includes 4 non-static member functions for recording a series of screenshot images: `RecordDesktop`, `RecordByTitle`, and 2 versions of `RecordByHandle` (Fig. 8).

53	// **** Non-Static Methods - Screen Record ****			
54				
55	<code>bool</code>	<code>RecordDesktop</code>	<code>(std::string</code>	<code>fileNameBase = "ScreenRecorder.png",</code>
56			<code>std::string</code>	<code>saveAsPath = "");</code>
57	<code>bool</code>	<code>RecordByTitle</code>	<code>(std::string</code>	<code>title,</code>
58			<code>std::string</code>	<code>fileNameBase = "ScreenRecorder.png",</code>
59			<code>std::string</code>	<code>saveAsPath = "");</code>
60	<code>bool</code>	<code>RecordByHandle</code>	<code>(std::string</code>	<code>handle,</code>
61			<code>std::string</code>	<code>fileNameBase = "ScreenRecorder.png",</code>
62			<code>std::string</code>	<code>saveAsPath = "");</code>
63	<code>bool</code>	<code>RecordByHandle</code>	<code>(HWND</code>	<code>hWnd,</code>
64			<code>std::string</code>	<code>fileNameBase = "ScreenRecorder.png",</code>
65			<code>std::string</code>	<code>saveAsPath = "");</code>

Fig. 8 Non-static member function (screen record) declarations in `ScreenRecorder.h`. See Appendix C for full documentation of `ScreenRecorder.h` and appendix D for full class implementation in `ScreenRecorder.cc`.

4.1 RecordDesktop

The `RecordDesktop` function saves a series of images of the desktop. `RecordDesktop` takes up to 2 string parameters.

If no parameters are supplied, a `ScreenRecorder` directory is created, if it does not already exist, in the same directory as the executable. The images of the desktop are saved as `N_ScreenRecorder.png` in this directory where `N` is the image number starting from 0 (Fig. 9, line 68).


```

65 // **** ScreenRecorder::RecordDesktop *****
66
67 // RecordDesktop
68 if (!recorder1.RecordDesktop()) {
69     std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
70 } // if (!recorder1.RecordDesktop())
71
72 // RecordDesktop(saveAsFile)
73 if (!recorder2.RecordDesktop("RecordDesktop.bmp")) {
74     std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
75 } // if (!recorder2.RecordDesktop(...))
76
77 // RecordDesktop(saveAsFile, saveAsPath)
78 if (!recorder3.RecordDesktop("RecordDesktop.gif",
79                             "C:\\Documents\\ScreenRecorder")) {
80     std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
81 } // if (!recorder3.RecordDesktop(...))

```

Fig. 9 Sample usage of RecordDesktop functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.

If a single string parameter is supplied, this parameter specifies the base of the save-as filename. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the same directory as the executable. The images of the desktop are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 9, line 73).

If 2 string parameters are supplied, the first parameter specifies the base of the save-as filename and the second is the path to the desired save location. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the specified directory. The images of the desktop are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 9, lines 78 and 79).

The return value of RecordDesktop indicates whether or not the desired series of images of the desktop were successfully saved. If the user specifies the filename base, the RecordDesktop function will return false if the filename base does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, RecordDesktop will return false.

4.2 RecordByTitle

The RecordByTitle function saves a series of images of the window identified by the specified title. RecordByTitle requires at least one string parameter (the title of the window to capture) followed by up to 3 string parameters.

If only the required title string parameter is supplied, a ScreenRecorder directory is created, if it does not already exist, in the same directory as the executable. The images of the specified window are then saved as N_ScreenRecorder.png in this directory where N is the image number starting from 0 (Fig. 10, line 86).

If a single string parameter is supplied in addition to the required title string parameter, this parameter specifies the base of the save-as filename. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the same directory as the executable. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0, and filenameBase is the user-specified filename base (Fig. 10, lines 91–92).

If 2 string parameters are supplied in addition to the required title string parameter, the first parameter specifies the base of the save-as filename and the second is the path to the desired save location. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the specified directory. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0, and filenameBase is the user-specified filename base (Fig. 10, lines 97–99).

```

83 // **** ScreenRecorder::RecordByTitle ****
84
85 // RecordByTitle(windowTitle)
86 if (!recorder1.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]")) {
87     std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
88 } // if (!recorder1.RecordByTitle(...))
89
90 // RecordByTitle(windowTitle, saveAsFile)
91 if (!recorder2.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]",
92                             "RecordByTitle.jpeg")) {
93     std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
94 } // if (!recorder2.RecordByTitle(...))
95
96 // RecordByTitle(windowTitle, saveAsFile, saveAsPath)
97 if (!recorder3.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]",
98                             "RecordByTitle.png",
99                             "C:\\Documents\\ScreenRecorder")) {
100     std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
101 } // if (!recorder3.RecordByTitle(...))

```

Fig. 10 Sample usage of RecordByTitle functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.

The return value of RecordByTitle indicates whether or not the specified window was found and the desired series of images of this window were successfully saved. If the user specifies the filename base, the RecordByTitle function will return false if the filename base does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, RecordByTitle will return false.

As in CaptureByTitle, the “title” that appears at the top of a window (in the title bar) is not necessarily the actual title of the window. The easiest way to determine the actual title of the window is to open Windows Task Manager (ctrl + shift + esc), select the Applications tab, and identify the task that you are attempting to record. The task names are the actual window titles.

4.3 RecordByHandle

Both RecordByHandle functions save a series of images of the window identified by the specified window handle. Both RecordByHandle functions require one parameter specifying the handle to the window to record followed by up to 2 optional string parameters. The difference between the 2 RecordByHandle functions is the data type of the required parameter specifying the handle to the window to record. This parameter can either be a hex string identifier to the handle (e.g, "0012079E") or an HWND.

If only the required handle identifier parameter is supplied, a ScreenRecorder directory is created, if it does not already exist, in the same directory as the executable. The images of the specified window are then saved as N_ScreenRecorder.png in this directory where N is the image number starting from 0 (Fig. 11, lines 112 and 113).

```
103 // **** ScreenRecorder::RecordByHandle ****
104
105 // Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
106 HWND handleObject      = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");
107
108 // Hex value of handleObject saved as a string
109 std::string handleString = toString(handleObject);
110
111 // RecordByHandle(windowHandle)
112 if (!recorder1.RecordByHandle(handleObject)           ||
113     !recorder1.RecordByHandle(handleString)) {
114     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
115 } // if (!recorder1.RecordByHandle(...))
116
117 // RecordByHandle(windowHandle, saveAsFile)
118 if (!recorder2.RecordByHandle(handleObject, "RecordByHandle.tif") ||
119     !recorder2.RecordByHandle(handleString, "RecordByHandle.bmp")) {
120     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
121 } // if (!recorder2.RecordByHandle(...))
122
123 // RecordByHandle(windowHandle, saveAsFile, saveAsPath)
124 if (!recorder3.RecordByHandle(handleObject, "RecordByHandle.gif",
125                                 "C:\\Documents\\ScreenRecorder") ||
126     !recorder3.RecordByHandle(handleString, "RecordByHandle.jpeg",
127                                 "C:\\Documents\\ScreenRecorder")) {
128     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
129 } // if (!recorder3.RecordByHandle(...))
```

Fig. 11 Sample usage of RecordByHandle functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.

If a single string parameter is supplied in addition to the required handle identifier parameter, this parameter specifies the base of the save-as filename. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the same directory as the executable. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 11, lines 118 and 119).

If 2 string parameters are supplied in addition to the required handle identifier parameter, the first parameter specifies the base of the save-as filename and the second is the path to the desired save location. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the specified directory. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 11, lines 124–127).

The return value of RecordByHandle indicates whether or not the specified window was found and the desired series of images of this window were successfully saved. If the user specifies the filename base, the RecordByHandle function will return false if the filename base does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, RecordByHandle will return false.

5. Conclusion

The ScreenRecorder utility was developed for saving series of desktop or window screenshots directly to file on any system running Microsoft Windows as an alternative to restrictive third-party screen recording software options. With the use of Microsoft Visual Studio, the ScreenRecorder utility was developed as a C++ class that can be compiled as a library (static or dynamic) to be linked into a project or can easily be compiled into an executable with the use of wrapper code. One major advantage of structuring the ScreenRecorder utility this way and distributing the source code (as opposed to only distributing an executable) is that the ScreenRecorder can be directly embedded into and manipulated within new and existing models.

The ScreenRecorder utility has been tested on and is compatible with Microsoft Vista, 7, and 8 and Visual Studio Express 2008, 2010, and 2012. The utility supports BMP, GIF, JPEG, PNG, and TIFF image formats. Once the ScreenRecorder utility has saved a series of screenshots in the desired format, Windows Movie Maker (or other appropriate software) can be used to combine the screenshot images into a WMV or AVI movie file.

Examples of wrapper codes and linking options are provided to the user in Appendices A and B. All source code is provided to the user in Appendices C and D.

6. References and Notes

1. Microsoft Windows. [accessed 2014 Sep 2]. <http://windows.microsoft.com/en-us/windows/home>.
2. Visual Studio Express. [accessed 2014 Sep 2]. <http://msdn.microsoft.com/en-us/vstudio/aa718325.aspx>.
3. Microsoft Windows Movie Maker. [accessed 2014 Sep 2]. <http://windows.microsoft.com/en-us/windows-live/movie-maker>.
4. Microsoft Window Movie Maker tutorial. [accessed 2014 Sep 2]. <http://windows.microsoft.com/en-us/windows-vista/getting-started-with-windows-movie-maker>.
5. Although it is not OEM software, MATLAB provides another easy way to combine the screenshots into an AVI movie. See Appendix E for more details.
6. All string parameters are string objects representing sequences of characters as specified in International Standard Organization (ISO) International Standard ISO/IEC 14882:2012(E) – Programming Language C++, and defined in the <string> header file. [accessed 2014 Sep 2]. <https://isocpp.org/std/the-standard>.
7. In computing, a handle is a data type that represents and provides access to a resource loaded in memory.
8. Microsoft Windows HWND window handle [accessed 2014 Sep 2]. <http://msdn.microsoft.com/en-us/library/windows/desktop/aa383751%28v=vs.85%29.aspx>.
9. The frame rate in this constructor is specified as an integer, but the frame rate of a ScreenRecorder object will have type double. This was done to allow for 2 overloaded constructors, in which case both constructors could not take the same sequence of parameter types [i.e., ScreenRecorder(double, double)]. This restricts the value of the frame rate when setting it through the constructor. If a noninteger frame rate is desired, the user can either calculate the corresponding time step and use the appropriate constructor or set the frame rate using the SetFrameRate mutator discussed in Section 3.3.
10. When error checking for equality in parameter values, the ScreenRecorder utility uses a tolerance of 1.0e-10.

INTENTIONALLY LEFT BLANK.

Appendix A. ScreenRecorderExamples_Capture.cc

This appendix appears in its original form, without editorial change.

Examples of using screen capture functions.

```
1 // *****
2 // *****
3 // 12/01/2014 MKA *****
4 // ScreenRecorderExamples_Capture.cc: Examples of screen capture functions *****
5 // *****
6 // *****
7
8 #include <cstdio>
9 #include <iostream>
10 #include <sstream>
11 #include "ScreenRecorder.h"
12
13 // Link user32, gdi32, and gdiplus libraries
14 #pragma comment(lib, "user32")
15 #pragma comment(lib, "gdi32")
16 #pragma comment(lib, "gdiplus")
17
18 template < typename T >
19 std::string toString(T x) { // Converts x to a std::string
20     std::stringstream sstr;
21     sstr << x;
22     return sstr.str();
23 } // std::string toString
24
25 int main(int argc, char* argv[]) {
26     ScreenRecorder recorder; // ScreenRecorder object
27
28     // **** ScreenRecorder::CaptureDesktop *****
29
30     // CaptureDesktop
31     if (!ScreenRecorder::CaptureDesktop() || // Invoked without use of a class object
32         !recorder.CaptureDesktop()) { // Invoked through a class object
33         std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
34     } // if (!CaptureDesktop())
35
36     // CaptureDesktop(saveAsFile)
37     if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.bmp") ||
38         !recorder.CaptureDesktop("CaptureDesktop.bmp")) {
39         std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
40     } // if (!CaptureDesktop(...))
41
42     // CaptureDesktop(saveAsFile, saveAsPath)
43     if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.gif",
44         "C:\\Documents\\ScreenRecorder") ||
45         !recorder.CaptureDesktop("CaptureDesktop.gif",
46         "C:\\Documents\\ScreenRecorder")) {
47         std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
48     } // if (!CaptureDesktop(...))
49
50     // **** ScreenRecorder::CaptureByTitle *****
51
52     // CaptureByTitle(windowTitle)
53     if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]") ||
54         !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]")) {
55         std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
56     } // if (!CaptureByTitle(...))
57
58     // CaptureByTitle(windowTitle, saveAsFile)
59     if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
```



```

60         "CaptureByTitle.jpeg") ||
61         !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
62         "CaptureByTitle.jpg")) {
63         std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
64     } // if (!CaptureByTitle(...))
65
66     // CaptureByTitle(windowTitle, saveAsFile, saveAsPath)
67     if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
68     "CaptureByTitle.png",
69     "C:\\Documents\\ScreenRecorder") ||
70     !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
71     "CaptureByTitle.png",
72     "C:\\Documents\\ScreenRecorder")) {
73         std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
74     } // if (!CaptureByTitle(...))
75
76     // **** ScreenRecorder::CaptureByHandle *****
77
78     // Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
79     HWND handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");
80
81     // Hex value of handleObject saved as a string
82     std::string handleString = toString(handleObject);
83
84     // CaptureByHandle(windowHandle)
85     if (!ScreenRecorder::CaptureByHandle(handleObject) ||
86     !recorder.CaptureByHandle(handleObject) ||
87     !ScreenRecorder::CaptureByHandle(handleString) ||
88     !recorder.CaptureByHandle(handleString)) {
89         std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
90     } // if (!CaptureByHandle(...))
91
92     // CaptureByHandle(windowHandle, saveAsFile)
93     if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.tiff") ||
94     !recorder.CaptureByHandle(handleObject, "CaptureByHandle.tif") ||
95     !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.bmp") ||
96     !recorder.CaptureByHandle(handleString, "CaptureByHandle.gif")) {
97         std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
98     } // if (!CaptureByHandle(...))
99
100    // CaptureByHandle(windowHandle, saveAsFile, saveAsPath)
101    if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.jpg",
102    "C:\\Documents\\ScreenRecorder") ||
103    !recorder.CaptureByHandle(handleObject, "CaptureByHandle.png",
104    "C:\\Documents\\ScreenRecorder") ||
105    !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.tif",
106    "C:\\Documents\\ScreenRecorder") ||
107    !recorder.CaptureByHandle(handleString, "CaptureByHandle.bmp",
108    "C:\\Documents\\ScreenRecorder")) {
109        std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
110    } // if (!CaptureByHandle(...))
111
112    system("pause");
113
114    return 0;
115 } // int main

```

INTENTIONALLY LEFT BLANK.

Appendix B. ScreenRecorderExamples_Record.cc

This appendix appears in its original form, without editorial change.

Examples of using screen recording functions.

```
1 // *****
2 // *****
3 // 12/01/2014 MKA *****
4 // ScreenRecorderExamples_Record.cc: Examples of screen recording functions *****
5 // *****
6 // *****
7
8 #include <cstdio>
9 #include <iostream>
10 #include <sstream>
11 #include "ScreenRecorder.h"
12
13 // Link user32, gdi32, and gdiplus libraries
14 #pragma comment(lib, "user32")
15 #pragma comment(lib, "gdi32")
16 #pragma comment(lib, "gdiplus")
17
18 template < typename T >
19 std::string toString(T x) { // Converts x to a std::string
20     std::stringstream sstr;
21     sstr << x;
22     return sstr.str();
23 } // std::string toString
24
25 int main(int argc, char* argv[]) {
26
27     // **** ScreenRecorder Object Constructors *****
28
29     ScreenRecorder recorder1; // Default constructor
30     ScreenRecorder recorder2(60.0, 10); // Specifying frame rate and recording time
31     ScreenRecorder recorder3(60.0, 1.0); // Specifying time step and recording time
32
33     // **** ScreenRecorder Getters and Setters *****
34
35     std::cout << "Default record time (s) = " << recorder1.GetRecordTime() << '\n';
36     std::cout << "Default frame rate (fps) = " << recorder1.GetFrameRate() << '\n';
37     std::cout << "Default time step (s) = " << recorder1.GetTimeStep() << '\n';
38     std::cout << std::endl;
39     // Output:
40     // Default record time (s) = 60.0
41     // Default frame rate (fps) = 1.0
42     // Default time step (s) = 1.0
43
44     recorder1.SetFrameRate(2); // Also changes time step!
45     std::cout << "Default record time (s) = " << recorder1.GetRecordTime() << '\n';
46     std::cout << "New frame rate (fps) = " << recorder1.GetFrameRate() << '\n';
47     std::cout << "New time step (s) = " << recorder1.GetTimeStep() << '\n';
48     std::cout << std::endl;
49     // Output:
50     // Default record time (s) = 60.0
51     // New frame rate (fps) = 2.0
52     // New time step (s) = 0.5
53
54     recorder1.SetRecordTime(120.0);
55     recorder1.SetTimeStep(2.0); // Also changes frame rate!
56     std::cout << "New record time (s) = " << recorder1.GetRecordTime() << '\n';
57     std::cout << "New frame rate (fps) = " << recorder1.GetFrameRate() << '\n';
58     std::cout << "New time step (s) = " << recorder1.GetTimeStep() << '\n';
59     std::cout << std::endl;
```

```

60 // Output:
61 // New record time (s)      = 120.0
62 // New frame rate (fps)    = 0.5
63 // New time step (s)      = 2.0
64
65 // **** ScreenRecorder::RecordDesktop *****
66
67 // RecordDesktop
68 if (!recorder1.RecordDesktop()) {
69     std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
70 } // if (!recorder1.RecordDesktop())
71
72 // RecordDesktop(fileNameBase)
73 if (!recorder2.RecordDesktop("RecordDesktop.bmp")) {
74     std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
75 } // if (!recorder2.RecordDesktop(...))
76
77 // RecordDesktop(fileNameBase, saveAsPath)
78 if (!recorder3.RecordDesktop("RecordDesktop.gif",
79                             "C:\\Documents\\ScreenRecorder")) {
80     std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
81 } // if (!recorder3.RecordDesktop(...))
82
83 // **** ScreenRecorder::RecordByTitle *****
84
85 // RecordByTitle(windowTitle)
86 if (!recorder1.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]")) {
87     std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
88 } // if (!recorder1.RecordByTitle(...))
89
90 // RecordByTitle(windowTitle, fileNameBase)
91 if (!recorder2.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]",
92                             "RecordByTitle.jpeg")) {
93     std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
94 } // if (!recorder2.RecordByTitle(...))
95
96 // RecordByTitle(windowTitle, fileNameBase, saveAsPath)
97 if (!recorder3.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]",
98                             "RecordByTitle.png",
99                             "C:\\Documents\\ScreenRecorder")) {
100     std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
101 } // if (!recorder3.RecordByTitle(...))
102
103 // **** ScreenRecorder::RecordByHandle *****
104
105 // Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
106 HWND handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");
107
108 // Hex value of handleObject saved as a string
109 std::string handleString = toString(handleObject);
110
111 // RecordByHandle(windowHandle)
112 if (!recorder1.RecordByHandle(handleObject) ||
113     !recorder1.RecordByHandle(handleString)) {
114     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
115 } // if (!recorder1.RecordByHandle(...))
116
117 // RecordByHandle(windowHandle, fileNameBase)
118 if (!recorder2.RecordByHandle(handleObject, "RecordByHandle.tif") ||
119     !recorder2.RecordByHandle(handleString, "RecordByHandle.bmp")) {
120     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';

```

```

121 } // if (!recorder2.RecordByHandle(...))
122
123 // RecordByHandle(windowHandle, fileNameBase, saveAsPath)
124 if (!recorder3.RecordByHandle(handleObject, "RecordByHandle.gif",
125     "C:\\Documents\\ScreenRecorder") ||
126     !recorder3.RecordByHandle(handleString, "RecordByHandle.jpeg",
127     "C:\\Documents\\ScreenRecorder")) {
128     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
129 } // if (!recorder3.RecordByHandle(...))
130
131 system("pause");
132
133 return 0;
134 } // int main

```

Appendix C. ScreenRecorder.h

This appendix appears in its original form, without editorial change.

Interface for the ScreenRecorder class.

```
1 // *****
2 // *****
3 // 12/01/2014 MKA *****
4 // ScreenRecorder.h: Interface for the ScreenRecorder class. *****
5 // *****
6 // *****
7
8 #ifndef _SCREEN_RECORDER_H_
9 #define _SCREEN_RECORDER_H_
10 #include <sstream>
11 #include <string>
12 #include <windows.h>
13
14 #define TOL 1.0e-10
15 #define DEFAULT_MAXT 60.0
16 #define DEFAULT_DT 1.0
17 #define DEFAULT_FPS 1
18
19 class ScreenRecorder {
20 public:
21 // **** Static Methods - Screen Capture *****
22
23 static bool CaptureDesktop(std::string saveAsFile = "ScreenCapture.png",
24                             std::string saveAsPath = "");
25 static bool CaptureByTitle(std::string title,
26                             std::string saveAsFile = "ScreenCapture.png",
27                             std::string saveAsPath = "");
28 static bool CaptureByHandle(std::string handle,
29                             std::string saveAsFile = "ScreenCapture.png",
30                             std::string saveAsPath = "");
31 static bool CaptureByHandle(HWND hWnd,
32                             std::string saveAsFile = "ScreenCapture.png",
33                             std::string saveAsPath = "");
34
35 // **** Constructors *****
36
37 ScreenRecorder();
38 ScreenRecorder(double recordTime, int frameRate);
39 ScreenRecorder(double recordTime, double timeStep);
40
41 // **** Accessors *****
42
43 double GetRecordTime();
44 double GetFrameRate();
45 double GetTimeStep();
46
47 // **** Mutators *****
48
49 void SetRecordTime(double recordTime);
50 void SetFrameRate(double frameRate);
51 void SetTimeStep(double dT);
52
53 // **** Non-Static Methods - Screen Record *****
54
55 bool RecordDesktop(std::string fileNameBase = "ScreenRecorder.png",
56                    std::string saveAsPath = "");
57 bool RecordByTitle(std::string title,
58                    std::string fileNameBase = "ScreenRecorder.png",
59                    std::string saveAsPath = "");
```



```

60     bool        RecordByHandle(std::string handle,
61                                std::string fileNameBase = "ScreenRecorder.png",
62                                std::string saveAsPath    = "");
63     bool        RecordByHandle(HWND hWnd,
64                                std::string fileNameBase = "ScreenRecorder.png",
65                                std::string saveAsPath    = "");
66
67 private:
68     // **** Class Variables ****
69
70     double fps; // framerate (frames per second)
71     double dT;  // timestep (sec)
72     double maxT; // total time to record (sec)
73
74     // **** Private Methods ****
75
76     static std::string GetExeFileName();
77     static std::string GetExePath();
78     static wchar_t* GetWC(const char *c);
79     static int GetEncoderClsid(const WCHAR* format, CLSID* pClsid);
80
81     template < typename T >
82     static std::string toLower(T str) { // Changes str to all lower case std::string
83         unsigned int i;
84         std::string newStr = static_cast<std::string>(str);
85         for (i = 0; i < newStr.length(); ++i) {
86             newStr[i] = tolower(str[i]);
87         } // for (i = 0; i < newStr.length(); ++i)
88         return newStr;
89     } // std::string toLower
90
91     template < typename T >
92     std::string toString(T x) { // Converts x to a std::string
93         std::stringstream sstr;
94         sstr << x;
95         return sstr.str();
96     } // std::string toString
97 }; // class ScreenRecorder
98 #endif // #ifndef _SCREEN_RECORDER_H_

```

INTENTIONALLY LEFT BLANK.

Appendix D. ScreenRecorder.cc

This appendix appears in its original form, without editorial change.

Implementation of the ScreenRecorder class.

```
1 // *****
2 // *****
3 // 12/01/2014 MKA *****
4 // ScreenRecorder.cc: Implementation of the ScreenRecorder class. *****
5 // *****
6 // *****
7
8 #define _CRT_SECURE_NO_DEPRECATED
9 #include <ctime>
10 #include <direct.h>
11 #include <errno.h>
12 #include <windows.h> // Has to be before <gdiplus.h>
13 #include <gdiplus.h>
14 #include <iostream>
15 #include <string>
16 #include "ScreenRecorder.h"
17
18 // *****
19 // **** Static Methods - Screen Capture *****
20 // *****
21
22 bool ScreenRecorder::CaptureDesktop(std::string saveAsFile,
23                                     std::string saveAsPath) {
24     return CaptureByHandle(NULL, saveAsFile, saveAsPath);
25 } // bool ScreenRecorder::CaptureDesktop
26
27 bool ScreenRecorder::CaptureByTitle(std::string title,
28                                     std::string saveAsFile,
29                                     std::string saveAsPath) {
30     HWND hWnd = FindWindow(NULL, TEXT(title.c_str()));
31     if (!hWnd) {
32         return false;
33     } // if (!hWnd)
34     return CaptureByHandle(hWnd, saveAsFile, saveAsPath);
35 } // bool ScreenRecorder::CaptureByTitle
36
37 bool ScreenRecorder::CaptureByHandle(std::string handle,
38                                     std::string saveAsFile,
39                                     std::string saveAsPath) {
40     HWND hWnd = (HWND) wcstoul(GetWC(handle.c_str()), NULL, 16);
41     if (!hWnd) {
42         return false;
43     } // if (!hWnd)
44     return CaptureByHandle(hWnd, saveAsFile, saveAsPath);
45 } // bool ScreenRecorder::CaptureByHandle
46
47 bool ScreenRecorder::CaptureByHandle(HWND hWnd,
48                                     std::string saveAsFile,
49                                     std::string saveAsPath) {
50     Gdiplus::GdiplusStartupInput gdiplusStartupInput;
51     ULONG_PTR gdiplusToken;
52     Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
53
54     RECT rect = {0};
55     int width = GetSystemMetrics(SM_CXSCREEN);
56     int height = GetSystemMetrics(SM_CYSCREEN);
57     if (saveAsPath.length() == 0){
58         saveAsPath = GetExePath();
59     } // if (saveAsPath.length() == 0)
```

```

60 if (saveAsPath[saveAsPath.length()-1] != '\\') {
61     saveAsPath += '\\';
62 } // if (saveAsPath[saveAsPath.length()-1] != '\\')
63 wchar_t* fileName = GetWC((saveAsPath+saveAsFile).c_str());
64 std::string extension = saveAsFile.substr(saveAsFile.find_last_of('.') + 1,
65                                           saveAsFile.length());
66
67 extension = toLower(extension);
68 if (extension == "jpg") {
69     extension = "jpeg";
70 } else if (extension == "tif") {
71     extension = "tiff";
72 } // adjust extension to account for common alternate format abbreviations
73 wchar_t* format = GetWC(("image/" + extension).c_str());
74
75 HWND desktop = GetDesktopWindow();
76 HDC desktopHdc = GetDC(desktop);
77 HDC destHdc = CreateCompatibleDC(desktopHdc);
78 HDC iconHdc = CreateCompatibleDC(desktopHdc);
79
80 if (hWnd) {
81     // If window handle was specified, then:
82     // 1) bring window to front
83     // 2) fix the window to stay on top and not change size or position
84     // 3) update capture region size
85     // Otherwise, capture the entire desktop
86     if (IsIconic(hWnd)) {
87         ShowWindow(hWnd, SW_SHOWNORMAL);
88     } // if (IsIconic(hWnd))
89     SetWindowPos(hWnd, HWND_TOPMOST, 0, 0, 0, 0,
90                 SWP_NOMOVE | SWP_NOSIZE | SWP_SHOWWINDOW);
91     GetWindowRect(hWnd, &rect);
92     // Double check that the window is valid, if not, then capture entire desktop
93     if (rect.right >= rect.left) {
94         width = rect.right - rect.left;
95     } // if (rect.right >= rect.left)
96     if (rect.bottom >= rect.top) {
97         height = rect.bottom - rect.top;
98     } // if (rect.bottom >= rect.top)
99 } // if (hWnd)
100
101 // Copy image to HBITMAP object
102 HBITMAP destBmp = CreateCompatibleBitmap(desktopHdc, width, height);
103 HBITMAP oldBmp = (HBITMAP)SelectObject(destHdc, destBmp);
104 BitBlt(destHdc, 0, 0, width, height,
105         desktopHdc, rect.left, rect.top, SRCCOPY | CAPTUREBLT);
106 SelectObject(destHdc, oldBmp);
107
108 // Prepare to save in specified format (bmp, jpeg, gif, tiff, or png)
109 Gdiplus::Bitmap* newBmp = Gdiplus::Bitmap::FromHBITMAP(destBmp, NULL);
110 Gdiplus::Status status = Gdiplus::GenericError;
111 CLSID clsid;
112 if (newBmp && (GetEncoderClsid(format, &clsid) != -1)) {
113     // Able to create Bitmap from HBITMAP and found valid clsid
114     status = newBmp->Save(fileName, &clsid, NULL);
115 } // if (newBmp && (GetEncoderClsid(format, &clsid) != -1))
116
117 // Cleanup
118 if (hWnd) {
119     SetWindowPos(hWnd, HWND_NOTOPMOST, rect.left, rect.top, width, height,
120                 SWP_SHOWWINDOW);

```

```

121     } // if (hWdn)
122     if (newBmp) {
123         delete newBmp;
124     } // if (newBmp)
125     Gdiplus::GdiplusShutdown(gdiplusToken);
126     ReleaseDC(desktop, desktopHdc);
127     DeleteObject(destBmp);
128     DeleteDC(destHdc);
129
130     return (status == Gdiplus::Ok);
131 } // bool ScreenRecorder::CaptureByHandle
132
133 // *****
134 // **** Constructors *****
135 // *****
136
137 ScreenRecorder::ScreenRecorder() {
138     SetRecordTime(DEFAULT_MAXT);
139     SetTimeStep(DEFAULT_DT);
140 } // ScreenRecorder::ScreenRecorder
141
142 ScreenRecorder::ScreenRecorder(double recordTime, int frameRate) {
143     SetRecordTime(recordTime);
144     SetFrameRate(static_cast<double>(frameRate));
145 } // ScreenRecorder::ScreenRecorder
146
147 ScreenRecorder::ScreenRecorder(double recordTime, double timeStep) {
148     SetRecordTime(recordTime);
149     SetTimeStep(timeStep);
150 } // ScreenRecorder::ScreenRecorder
151
152 // *****
153 // **** Accessors *****
154 // *****
155
156 double ScreenRecorder::GetRecordTime() {
157     return maxT;
158 } // double ScreenRecorder::GetRecordTime
159
160 double ScreenRecorder::GetFrameRate() {
161     return fps;
162 } // double ScreenRecorder::GetFrameRate
163
164 double ScreenRecorder::GetTimeStep() {
165     return dT;
166 } // double ScreenRecorder::GetTimeStep
167
168 // *****
169 // **** Mutators *****
170 // *****
171
172 void ScreenRecorder::SetRecordTime(double recordTime) {
173     if (recordTime < TOL) {
174         std::cerr << "ScreenRecorder: Record time must be >= 0.0" << '\n';
175         std::cerr << "Continuing with default record time ("
176                 << DEFAULT_MAXT << " s)" << std::endl;
177         recordTime = DEFAULT_MAXT;
178     } // if (recordTime < 0.0)
179     maxT = recordTime;
180 } // void ScreenRecorder::SetRecordTime
181

```

```

182 void ScreenRecorder::SetFrameRate(double frameRate) {
183     if (frameRate <= 0.0) {
184         std::cerr << "ScreenRecorder: Frame rate must be > 0" << '\n';
185         std::cerr << "Continuing with default frame rate ("
186             << DEFAULT_FPS << ")" << std::endl;
187         frameRate = DEFAULT_FPS;
188     } // if (frameRate <= 0.0)
189     fps = frameRate;
190     dT = 1.0 / fps;
191 } // void ScreenRecorder::SetFrameRate
192
193 void ScreenRecorder::SetTimeStep(double timeStep) {
194     if (timeStep <= 0.0) {
195         std::cerr << "ScreenRecorder: Time step must be > 0.0" << '\n';
196         std::cerr << "Continuing with default time step ("
197             << DEFAULT_DT << " s)" << std::endl;
198         timeStep = DEFAULT_DT;
199     } // if (timeStep <= 0.0)
200     dT = timeStep;
201     fps = 1.0 / dT;
202 } // void ScreenRecorder::SetTimeStep
203
204 // *****
205 // **** Non-Static Members - Screen Record *****
206 // *****
207
208 bool ScreenRecorder::RecordDesktop(std::string fileNameBase,
209                                     std::string saveAsPath) {
210     return RecordByHandle(NULL, fileNameBase, saveAsPath);
211 } // bool ScreenRecorder::RecordDesktop
212
213 bool ScreenRecorder::RecordByTitle(std::string title,
214                                     std::string fileNameBase,
215                                     std::string saveAsPath) {
216     HWND hWnd = FindWindow(NULL, TEXT(title.c_str()));
217     if (!hWnd) {
218         return false;
219     } // if (!hWnd)
220     return RecordByHandle(hWnd, fileNameBase, saveAsPath);
221 } // bool ScreenRecorder::RecordByTitle
222
223 bool ScreenRecorder::RecordByHandle(std::string handle,
224                                     std::string fileNameBase,
225                                     std::string saveAsPath) {
226     HWND hWnd = (HWND) wcstoul(GetWC(handle.c_str()), NULL, 16);
227     if (!hWnd) {
228         return false;
229     } // if (!hWnd)
230     return RecordByHandle(hWnd, fileNameBase, saveAsPath);
231 } // bool ScreenRecorder::RecordByHandle
232
233 bool ScreenRecorder::RecordByHandle(HWND hWnd,
234                                     std::string fileNameBase,
235                                     std::string saveAsPath) {
236     clock_t endTick;
237     clock_t t1;
238     clock_t t2;
239     errno_t err;
240     int tickStep = static_cast<int>(dT * CLOCKS_PER_SEC);
241     int frame = 0;
242     int status;

```

```

243     bool        wasSuccessful = true;
244     std::string saveAsFile;
245
246     if (saveAsPath.length() == 0) {
247         saveAsPath = GetExePath();
248     } // if (saveAsPath.length() == 0)
249     if (saveAsPath[saveAsPath.length()-1] != '\\') {
250         saveAsPath += '\\';
251     } // if (saveAsPath[saveAsPath.length()-1] != '\\')
252     saveAsPath += fileNameBase.substr(0, fileNameBase.find_last_of('.'));
253     saveAsFile = "_" + fileNameBase;
254
255     _set_errno(0);
256     status = _mkdir(saveAsPath.c_str());
257     _get_errno(&err);
258     wasSuccessful = ((status == 0) || (err == EEXIST));
259     saveAsPath += "\\";
260
261     t1 = clock();
262     endTick = t1 + static_cast<int>(maxT * CLOCKS_PER_SEC);
263     t2 = t1;
264
265     while (wasSuccessful && (t2 < endTick)) {
266         t2 = clock();
267         if ((t2 - t1) > tickStep) {
268             wasSuccessful = ScreenRecorder::CaptureByHandle(hWnd,
269                                                         toString(frame) + saveAsFile, saveAsPath);
270             ++frame;
271             t1 = t2;
272         } // if ((t2 - t1) > tickStep)
273     } // while (wasSuccessful && (t2 < endTick))
274
275     return wasSuccessful;
276 } // bool ScreenRecorder::RecordByHandle
277
278 // *****
279 // **** Private Methods *****
280 // *****
281
282 std::string ScreenRecorder::GetExeFileName() {
283     char buffer[MAX_PATH];
284     GetModuleFileName( NULL, buffer, MAX_PATH );
285     return std::string(buffer);
286 } // std::string ScreenRecorder::GetExeFileName
287
288 std::string ScreenRecorder::GetExePath() {
289     std::string f = GetExeFileName();
290     return f.substr(0, f.find_last_of( "\\\" )+1);
291 } // std::string ScreenRecorder::GetExePath
292
293 wchar_t* ScreenRecorder::GetWC(const char *c) {
294     const size_t cSize = strlen(c)+1;
295     wchar_t* wc = new wchar_t[cSize];
296     mbstowcs(wc, c, cSize);
297
298     return wc;
299 } // wchar_t* ScreenRecorder::GetWC
300
301 // http://msdn.microsoft.com/en-us/library/windows/desktop/ms533843%28v=vs.85%29.aspx
302 int ScreenRecorder::GetEncoderClsid(const WCHAR* format, CLSID* pClsid) {
303     // bmp, jpeg, gif, tiff, png

```



```

304  UINT num    = 0; // Number of image encoders
305  UINT size   = 0; // Size of the image encoder array in bytes
306
307  Gdiplus::ImageCodecInfo* pImageCodecInfo = NULL;
308
309  Gdiplus::GetImageEncodersSize(&num, &size);
310  if (size == 0) {
311      return -1; // Failure
312  } // if (size == 0)
313
314  pImageCodecInfo = (Gdiplus::ImageCodecInfo*)(malloc(size));
315  if (pImageCodecInfo == NULL) {
316      return -1; // Failure
317  } // if (pImageCodecInfo == NULL)
318
319  GetImageEncoders(num, size, pImageCodecInfo);
320
321  for (UINT j = 0; j < num; ++j) {
322      if (wcscmp(pImageCodecInfo[j].MimeType, format) == 0) {
323          *pClsid = pImageCodecInfo[j].Clsid;
324          free(pImageCodecInfo);
325          return j; // Success
326      } // if (wcscmp(pImageCodecInfo[j].MimeType, format) == 0)
327  } // for (UINT j = 0; j < num; ++j)
328
329  free(pImageCodecInfo);
330  return -1; // Failure
331 } // int ScreenRecorder::GetEncoderClsid

```

INTENTIONALLY LEFT BLANK.

Appendix E. MATLAB Script for Creating AVI Movie from Screenshots

This appendix appears in its original form, without editorial change.

```

1 %% Sample script for creating an AVI movie from individual images
2
3 % Path to where images are saved
4 imagePath = 'C:\Documents\ScreenRecorder\';
5
6 % File base name
7 imageName = 'RecordByHandle.tif';
8
9 % Name of created video
10 movieName = 'SampleMovie.avi';
11
12 % Number of images to stitch together
13 numFrames = 50;
14
15 % Rate of playback for the video in frames per second
16 frameRate = 5;
17
18 % Create video writer object and set frame rate
19 writerObj = VideoWriter(strcat(imagePath, movieName));
20 writerObj.FrameRate = frameRate;
21
22 % Open video writer object for writing
23 open(writerObj);
24
25 % Write frames to video writer object
26 for k = 0 : numFrames-1
27     fileName = strcat(imagePath, sprintf('%d%s', k, imageName));
28     thisImage = imread(fileName);
29     writeVideo(writerObj, thisImage);
30 end
31
32 % Close video writer object
33 close(writerObj);

```

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIO LL
IMAL HRA MAIL & RECORDS MGMT

1 GOVT PRINTG OFC
(PDF) A MALHOTRA

1 DIR USARL
(PDF) RDRL WML A
M ARTHUR

INTENTIONALLY LEFT BLANK.